16.3. HTML Versus XHTML

The majority of HTML is completely compatible with XHTML, and this book is devoted to that majority. In this chapter, however, we talk about the minority: where the HTML 4.01 standard and the XHTML DTD differ. If you truly desire to create documents that are both HTML- and XHTML-compliant, you must heed the various warnings and caveats we outline in the following sections.

The biggest difference -- that's Difference with a capital D and that spells difficult -- is that writing XHTML documents requires much more discipline and attention to detail than even the most fastidious HTML author ever dreamed necessary. In W3C parlance, that means your documents must be impeccably "well-formed." Throughout the history of HTML -- and in this book -- authors have been encouraged to create well-formed documents, but you would have to break rank with the HTML standards for your documents to be considered well-formed by XML standards.

Nonetheless, your efforts to master XHTML will be rewarded with documents that are well-formed and a sense of satisfaction from playing by the new rules. You will truly benefit in the future, too: through XML, your documents will be able to appear in places you never dreamed would exist (mostly good places, we hope).

16.3.1. Correctly Nested Elements

One requirement of a well-formed XHTML document is that its elements are nested correctly. This isn't any different from HTML standards: simply close the markup elements in the order in which you opened them. If one element is within another, the end tag of the inner element must appear before the end tag of the outer element.

Hence, in the following well-formed XHTML segment, we end the italics tag before we end the bold one, because we'd started italicizing after we had started bolding the content:

```
<b>Close the italics tag <i>first</i></b>.
```

On the other hand, the following:

```
<b>Well-formed, this is <i>not!</b></i>
```

is not well-formed.

XHTML strictly enforces other nesting restrictions that have always been part of HTML but not always enforced. These restrictions are not formally part of the XHTML DTD; they are instead defined as part of the XHTML standard that is based upon the DTD.[85]

[85]This is hair-splitting within the XHTML standard. The XML standard has no mechanism to define which tags may not be placed within another tag. SGML, upon which XML is based, does have such a feature, but it was removed from XML to make the language easier to use and implement. As a result, these restrictions

are simply listed in an appendix of the XHTML standard instead of explicitly defined in the XHTML DTD.

Nesting restrictions include:

- The `<a>` tag cannot contain another `<a>` tag.
- The `<pre>` tag cannot contain `<img>`, `<object>`, `<big>`, `<small>`, `<sub>`, or `<sup>`.
- The `<button>` tag cannot contain `<input>`, `<select>`, `<textarea>`, `<label>`, `<button>`, `<form>`, `<fieldset>`, `<iframe>`, or `<isindex>`.
- The `<label>` tag cannot contain other `<label>` tags.
- The `<form>` tag cannot contain other `<form>` tags.

These restrictions apply to nesting at any level. For example, an `<a>` tag cannot contain any other `<a>` tags, or any tag that in turn contains an `<a>` tag.

16.3.2. End Tags

As we've documented throughout this book, any HTML tag that contains other tags or content has a corresponding end tag. However, one of the hallmarks of HTML (codified in the 4.01 standard) is that you may leave out the end tags if their presence can be inferred by the processing agent. This is why most of us HTML authors commonly leave out the `</p>` end tag between adjacent paragraphs. Also, lists and tables can be complicated to wade through and not having to visually stumble over all the `</li>`, `</td>`, `</th>`, and `</tr>` end tags certainly makes HTML a little clearer and easier to read.

This is not so for XHTML. Every tag that contains other tags or content must have a corresponding end tag present, correctly nested within the XHTML document. A missing end tag is an error and renders the document non-compliant.

16.3.3. Handling Empty Elements

In XML, and thus XHTML, every tag must have a corresponding end tag, even those aren't allowed to contain other tags or content. Accordingly, XHTML expects the line break to appear as `<br></br>` in your document. Ugh.

Fortunately, there is an acceptable alternative: include a slash before the closing brace of the tag to indicate its ending, as in `<br />`. If the tag has attributes, the slash comes after all the attributes, so that an image could be defined as:

```
<img src="kumquat.gif" />
```

While this notation may seem foreign and annoying to an HTML author, it actually serves a useful purpose. Any XHTML element that has no content can be written this way. Thus, an empty paragraph can be written as `<p />` and an empty table cell can be written as `<td />`. This is a handy way to mark empty table cells.

Clever as it may seem, writing empty tags in this abbreviated way may confuse HTML browsers.

So to avoid compatibility problems, you can fool the HTML browsers by placing a space before the forward slash in an empty element using the XHTML version of its end tag. For example, use `<br />` with its space between the `br` and `/`, instead of the XHTML equivalents `<br/>` or `<br></br>`. Table 16-1 contains all the empty HTML tags expressed in their acceptable XHTML (transitional DTD) forms.

Table 16-1. HTML Empty Tags in XHTML Format

`<area />`	`<img />`
`<base />`	`<input />`
`<basefont />`	`<isindex />`
`<br />`	`<link />`
`<col />`	`<meta />`
`<frame />`	`<param />`
`<hr />`	

16.3.4. Case Sensitivity

If you thought getting all those end tags in the right place and cleaning up the occasional nesting error would make writing XHTML documents difficult, hold on to your hat. XHTML is case-sensitive for *all* tag and attribute names. In an XHTML document, `<a>` and `<A>` are different tags; `src` and `SRC` are different attributes. So are `sRc` and `SrC` ! How forgiving HTML seems now.

The XHTML DTD defines all former HTML tags and attributes using lowercase letters. Uppercase tag or attribute names are not valid XHTML tags or attributes.

This can be a difficult situation for any author wishing to convert existing HTML documents into XHTML-compliant ones. Lots of web pages use uppercase tag and attribute names, to make them stand out from the surrounding lowercase content.

To become compliant, all those names must be converted to lowercase, even the ones you'd used in your CSS style sheet definitions. Fortunately, this kind of change is easily accomplished with various editing tools. And XHTML authoring systems should perform the conversion for you.

16.3.5. Quoted Attribute Values

And as if all those case-sensitive attribute names weren't aggravating enough, XHTML requires that every attribute value -- even the numeric ones -- be enclosed in double quotes. In HTML, you could quote anything your heart desired, but quotes were only required if the attribute value included whitespace or other special characters. To be XHTML-compliant, every attribute must be enclosed in quotes.

For example:

```
<table rows=3>
```

is wrong in XHTML. It is correctly written:

```
    <table rows="3">
```

16.3.6. Explicit Attribute Values

Within HTML, there are a small number of attributes that have no value. Instead, their mere presence within a tag causes that tag to behave differently. In general, these attributes represent a sort of on/off switch for the tag, like the `compact` attribute for the various list tags or the `ismap` attribute for the `<img>` tag.

In XHTML, every attribute must have a value. Those without values must now use their own names. Thus, `compact` in XHTML is correctly specified as `compact="compact"; checked` is now `checked="checked"` Each must contain the newly required attribute value enclosed in quotes. Table 16-2 contains a list of newly valued attributes.

Table 16-2. New XHTML Values for Value-Less HTML Attributes

`compact="compact"`	`disabled="disabled"`
`nowrap="nowrap"`	`readonly="readonly"`
`ismap="ismap"`	`multiple="multiple"`
`declare="declare"`	`selected="selected"`
`noshade="noshade"`	`noresize="noresize"`
`checked="checked"`	`defer="defer"`

Be aware that this new attribute value requirement may cause some old HTML browsers to ignore the attribute altogether. HTML 4.0-compliant browsers don't have that problem, so the majority of users won't notice any difference. There is no good solution to this problem other than distributing HTML 4.0-compliant browsers to the needy.

16.3.7. Handling Special Characters

XHTML is more sensitive than HTML is to the use of the `<` and `&` characters in JavaScript and CSS declarations within your documents. In HTML, you can avoid potential conflicts by enclosing your scripts and stylesheets in comments (`<!--` and `-->`). XML browsers, however, may simply remove all the contents of comments from your document, thereby deleting your hidden scripts and stylesheets.

To properly shield your special characters from XML browsers, enclose your styles or scripts in a CDATA section. This tells the XML browser that any characters contained within are plain old characters, without special meanings. For example:

```
    <script language="JavaScript">
```

```
<![CDATA[
   JavaScript here...
 ]]>
</script>
```

This doesn't solve the problem, though. HTML browsers ignore the contents of the CDATA XML tag, but honor the contents of comment-enclosed scripts and stylesheets, whereas XML browsers do just the opposite. We recommend that you put your scripts and styles in external files and reference them in your document with appropriate external links.

Special characters in attribute values are problematic in XHTML, as well. In particular, an ampersand within an attribute value should always be written using `&amp;` and not simply an `&` character. Similarly, play it safe and encode less-than and greater-than signs using their `&lt;` and `&gt;` entities. For example, while:

```
<img src=seasonings.gif alt="Salt & pepper">
```

is perfectly valid HTML, it must be written as:

```
<img src="seasonings.gif" alt="Salt &amp; pepper" />
```

to be compliant XHTML.

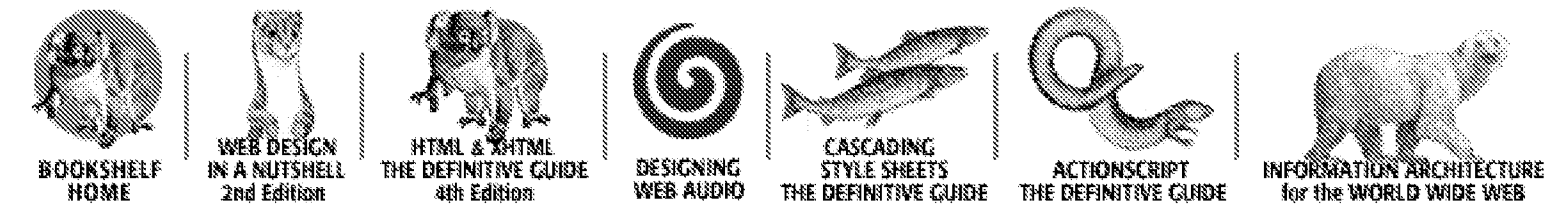
16.3.8. The id and name Attributes

Early versions of HTML used the `name` attribute with the `<a>` tag to create a fragment identifier in the document. This fragment could then be used in a URL to refer to a particular spot within a document. The `name` attribute was later added to other tags like `<frame>` and `<img>`, allowing those elements to also be referenced by name from other spots in the document.

With HTML 4.0, the W3C added the `id` attribute to almost every tag. Like `name`, `id` lets you associate an identifier with nearly any element in a document for later reference and use, perhaps by a hyperlink or a script.

XHTML has a strong preference for the `id` attribute as the anchor of choice within a document. The `name` attribute is defined, but formally deprecated for those elements that have historically used it. With widespread support of HTML 4.0 now in place, you should begin to avoid the `name` attribute where possible, and instead use the `id` attribute to bind names to elements in your documents. If you must use the `name` attribute on certain tags, include an identical `id` attribute to ensure that the tag will behave similarly when processed by an XHTML browser.

PREVIOUS | HOME | NEXT
16.2. Creating XHTML Documents | BOOK INDEX | 16.4. Should You Use XHTML?


BOOKSHELF HOME
WEB DESIGN IN A NUTSHELL 2nd Edition
HTML & XHTML THE DEFINITIVE GUIDE 4th Edition
DESIGNING WEB AUDIO
CASCADING STYLE SHEETS THE DEFINITIVE GUIDE
ACTIONSCRIPT THE DEFINITIVE GUIDE
INFORMATION ARCHITECTURE for the WORLD WIDE WEB